

Small Polygon Compression

Abhinav Jauhri, Martin Griss and Hakan Erdogmus

Carnegie Mellon University, ECE Department
Moffett Field, CA, 94035, USA

ajauhri@cmu.edu, martin.griss@sv.cmu.edu, hakan.erdogmus@sv.cmu.edu

Abstract

We introduce a compression technique for small polygons. The main application is to embed compressed polygons in emergency alert messages that have strict length restrictions, as in the case of Wireless Emergency Alert messages. We transform polygon coordinates to sets of integers and are able to compress them to between 10.4% and 25.6% of original length reducing original polygon lengths from 43-331 characters to 9-61 characters. The compression technique introduced in this work takes advantage of a strongly skewed polygon coordinate distribution. We also show that our technique is similar to and at times better than prior published state of the art integer compression techniques in terms of bits per integer.

1 Introduction

Geo-targeting is widely used on the Internet to improve user targeting with advertisements, multimedia content, information; essentially to improve the user experience and increase user engagement. Scenarios also exist where geo-targeting at a given time period becomes imperative for information exchange within a group or groups of people, thereby contributing to problems of network congestion and effectiveness. Emergency scenarios are a quintessential example where people in the affected area need to be informed and guided throughout the duration of an emergency. To address this, Wireless Emergency Alert (WEA) is a nation-wide service for broadcasting short messages* (currently 90 characters, similar to SMS messages) to all phones in a designated geographic area via activation of appropriate cell towers. The area is typically identified by a polygon as shown in Figure 1, though currently many operators use rather coarse-grained targeting (such as to a whole county).

In this work we study the compressibility of small polygons in the WEA message. This would enable a client side mobile application to filter the alert based on the location of the client, and thereby improve geo-targeting. We have available to us a corpus of 11,370 WEA messages sent out by the National Weather Service (NWS) [1][†]. The polygons in the NWS corpus range from 4-24 vertices, with a size ranging from 43-331 characters. Since WEA messages are broadcast to thousands of people, and it is believed that adding a polygon to more precisely define the target area is critical [2], it is essential to be able to compress typical WEA polygons to fit within the current message length, and also leave space for text of the alert.

*<https://www.fema.gov/wireless-emergency-alerts>

[†]These messages were sent out by the NWS in 2012, 2013, and 2014

We perform transformations to the original polygon coordinates to get a set of integers. These transformations are described in Section 3. Our compression technique *Bignum* is described in Section 4. We provide a comparative study of our technique with existing state of the art methods in Section 5.



Figure 1: A map showing 3 polygons (outlined by a yellow border). Actual broadcast of any alert covers a bigger area than shown by a polygon above.

2 Related Work

The compression problem we are tackling here is quite different from that described in published research on compression of polygon meshes [3–6] used by graphic applications. They typically are dealing with a large number of inter-connected polygons in a 2D or 3D representation of a surface or solid, and thus are compressing a large number of polygons at the same time. Many of these polygons share common points and edges, which can be exploited in the compression technique. In our case, we have a single, relatively small polygon to compress, and so can not amortize items such as a dictionary of common points.

Numerous integer compression techniques can be applied provided the vertices of the polygon have been transformed to a set of integers. The very earliest one due to Golomb [7] is known to be optimal for one-sided geometric distributions of non-negative integers [8]. A variation in selection of a parameter in Golomb encoding results in another technique called Rice encoding [9]. Microsoft’s Point Compression technique [10], used for geographic locations, is also motivated from Golomb. Coding techniques like Interpolative [11] cannot be applied to our problem since it is necessary for the ordering of the polygon coordinates to be maintained at the decoding step to be able to exactly recover the original polygon.

Recent work on integer compression focuses on computation speed and is based on applying vectorized schemes by exploiting SIMD instructions [12, 13]. For our problem the main issue is not speed but better lossless compression and to maintain the ordering of integers. Polygon coordinates, for our purposes, cannot be sorted, then compressed, and sent over to the client’s mobile for decoding.

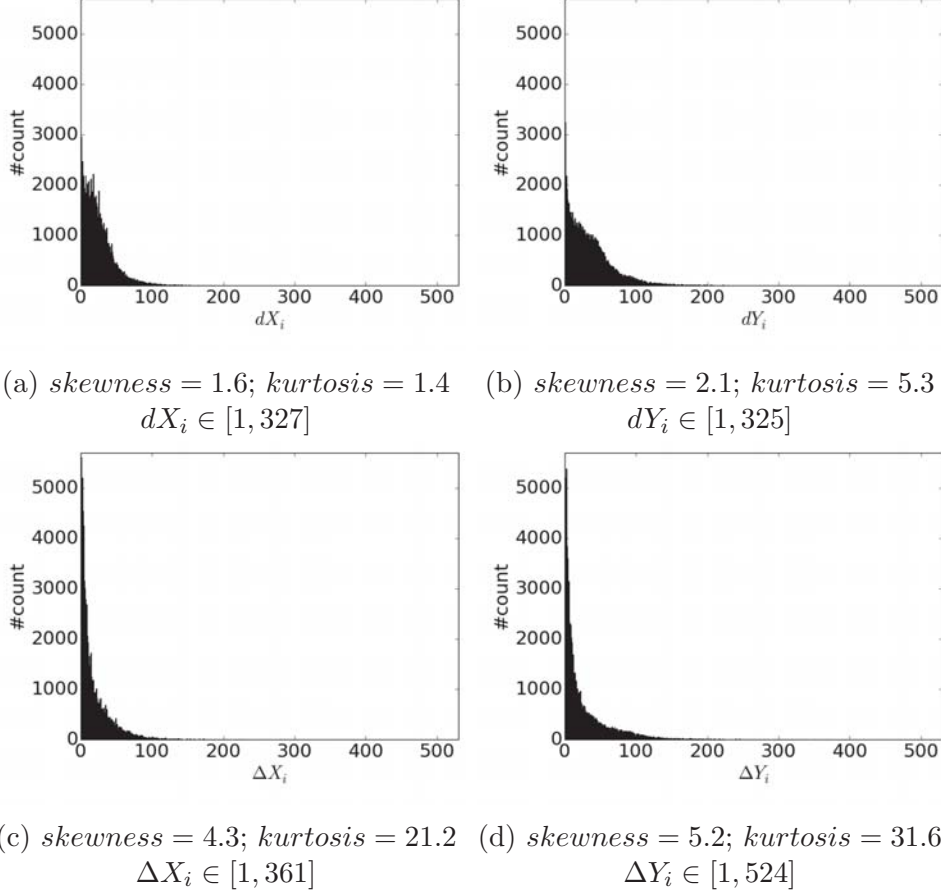


Figure 2: Positive skewness in both representations: $T^{\Delta^{min}}$ includes dX_i (2a) and dY_i (2b); and T^{Δ} includes ΔX_i (2c) and ΔY_i (2d). Heavy tails and peakedness define positive kurtosis [14]. Zero value counts are excluded from plots and measurements.

3 Polygon Transformations

In this section we describe the transformations to the original polygon which are necessary before application of any compression technique. We start with an original N point polygon, given as an ordered finite sequence in \mathbb{R}^2 :

$$O = \{X_1, Y_1, \dots, X_N, Y_N\}$$

where X_i, Y_i are GPS coordinates in decimal degrees.

In the NWS corpus, $N \in [4, 24]$, even though the NWS standard allows polygons of up to 100 points. The original uncompressed polygon length of 43 to 331 characters includes $2N - 1$ separating commas, $2N$ periods and N minus signs. X_i is typically $dd.dd$ and Y_i is typically $-dd.dd$ or $-ddd.dd$.

We perform transformation steps to the original set of coordinates O . The first transformation converts all coordinates to a set of positive integers, referred as O' . Using the set O' we create two representations of each polygon which will be used as the input set to the compression algorithms. The two representations are achieved

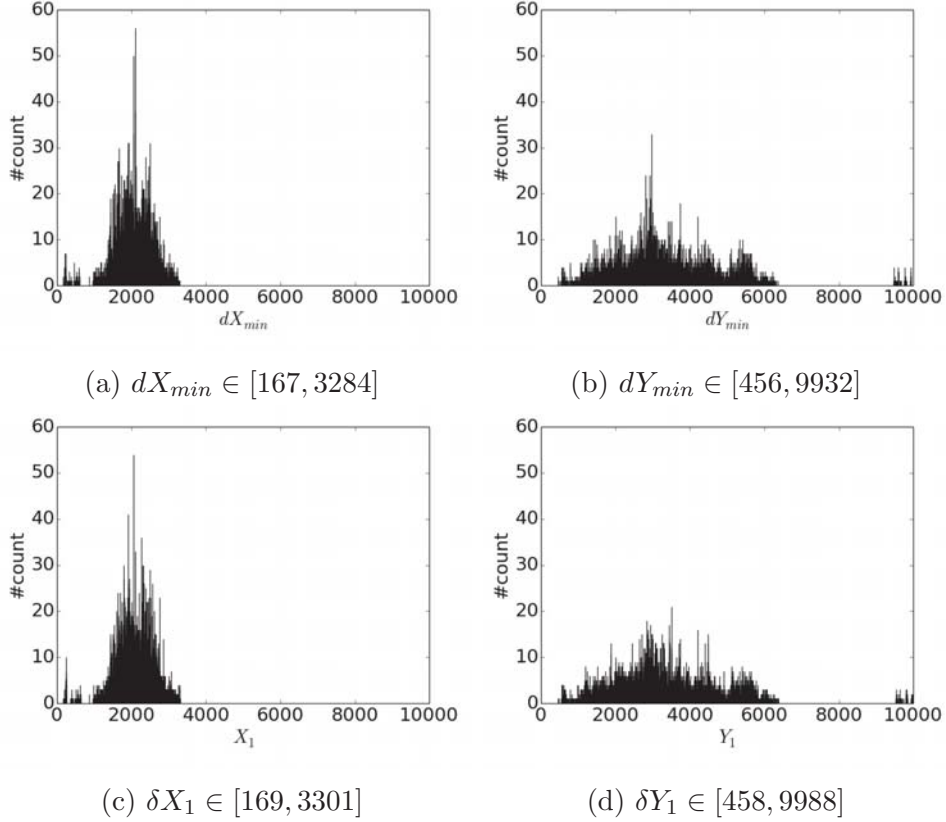


Figure 3: Ranges for large values in both transformations. $T^{\Delta_{min}}$ has (dX_{min}, dY_{min}) , and T^{Δ} has $(\delta X_1, \delta Y_1)$

by applying the following transformation rules:

1. *Delta Min representation:* Find the minimum x-coordinate and y-coordinate and take the difference with every vertex; the resultant set of integers is denoted as $T^{\Delta_{min}}$ and its elements are $\{dX_{min}, dY_{min}, dX_1, dY_1, \dots, dX_{N-1}, dY_{N-1}\}$.
2. *Delta representation:* Take the difference of consecutive coordinates; the resultant set is denoted as T^{Δ} and its elements are $\{\delta X_1, \delta Y_1, \Delta X_2, \Delta Y_2, \dots, \Delta X_{N-1}, \Delta Y_{N-1}\}$.

Details about both representations can be found in Appendix A.

T^{Δ} (popularly known as differential coding) representation is interesting, since the distribution (Figure 2) of its ΔX_i has a skewed and more peaked shape to that of the dX_i in $T^{\Delta_{min}}$, and also with a heavy tail. Simple transformations shown in Appendix A already produce a substantial compression because of the limited ranges and skewed distribution of the delta polygon coordinates (dX_i, dY_i) , and $(\Delta X_i, \Delta Y_i)$ and of the starting points (dX_{min}, dY_{min}) , and $(\delta X_1, \delta Y_1)$, shown in Figures 2, 3. It is important to note that in both representations we have two different sets of integers to compress, with distinctly different ranges and distributions: the single starting point pair of dX_{min} and dY_{min} for $T^{\Delta_{min}}$ (or δX_1 and δY_1 for T^{Δ}) and the $N - 1$ pairs of (dX_i, dY_i) for $T^{\Delta_{min}}$ (or $N - 2$ pairs of $(\Delta X_i, \Delta Y_i)$ for T^{Δ}).

4 Bignum Compression

Bignum compression starts by combining each delta pair (dX_i, dY_i) or $(\Delta X_i, \Delta Y_i)$ into a larger single number by the following algebraic expression:

$$dXY_i = dX_i * M + dY_i, \quad (1)$$

where M can be a fixed choice or chosen based on the specific range of dX_i and dY_i . For instance based on our corpus $M = 350$ will be an appropriate value to “make space” for dY_i because dY_i is always less than 350. The intuition behind equation 1 is to left shift bits intelligently such that there is always enough space to squeeze in dY_i . Likewise, we can combine (dX_{min}, dY_{min}) or $(\delta X_1, \delta Y_1)$ using a larger factor:

$$dXY_{min} = dX_{min} * Y_{factor} + dY_{min}, \quad (2)$$

where Y_{factor} can be chosen based on the range of dY_{min} . An appropriate value based on the corpus is 10,000 to make space for the dY_{min} .

Expanding on this combined pair of deltas concept, we can aggregate all deltas into a single big integer using arbitrary precision integer arithmetic libraries. The big integer is computed by successive pairing of elements of $T^{\Delta min}$ as shown in Figure 4:

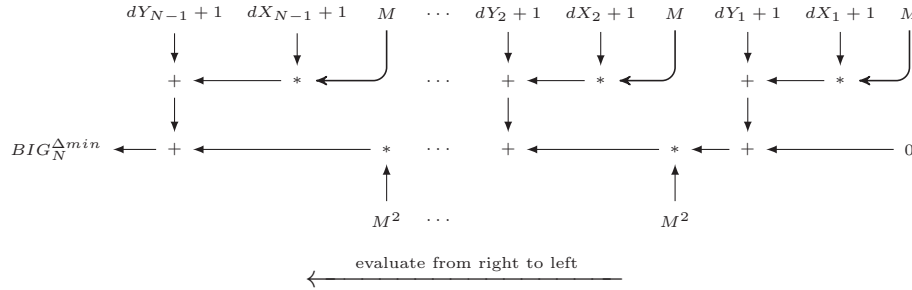


Figure 4: Sequence of operations for integers in $T^{\Delta min}$ to obtain a big integer $BIG_N^{\Delta min}$

In recursive form, the compression technique can be written as:

$$BIG_{i+1}^{\Delta min} = (BIG_i^{\Delta min} * M^2) + ((dX_i + 1) * M) + (dY_i + 1), \quad (3)$$

where $i \in [1, N - 1]$, and $BIG_1^{\Delta min}$ is 0. We add one to all dX_i and dY_i values to avoid the pathological case when delta values are zero.[‡] For the set T^{Δ} we use the same sequence of steps except that there is one pair less (Figure 5). The recursive formulation can be written as:

$$BIG_{i+1}^{\Delta} = (BIG_i^{\Delta} * M^2) + ((\Delta X_i + 1) * M) + (\Delta Y_i + 1), \quad (4)$$

where $i \in [2, N - 1]$ and BIG_2^{Δ} is 0.

[‡]Strictly speaking, only the first value, dX_1 could cause a problem.

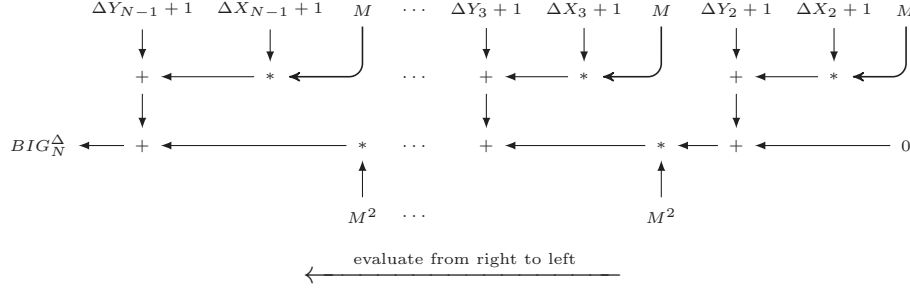


Figure 5: Sequence of operations for integers in T^Δ to obtain a big integer BIG_N^Δ

Rather than choosing a single value for M for the corpus, we ideally would choose a separate value for each polygon, defined by:

$$M = \max(\sup_i dX_i, \sup_i dY_i) + 2^\S \quad (5)$$

Larger values for the starting points in $T^{\Delta_{min}}$ and T^Δ could also be included in BIG by choosing an appropriate set of factors based on the distribution, such as $X_{factor} = 3500$ and $Y_{factor} = 10000$ for NWS corpus, and then apply the following:

$$BIG_{N+1}^{\Delta_{min}} = (BIG_N^{\Delta_{min}} * X_{factor} + dX_{min}) * Y_{factor} + dY_{min} \quad (6)$$

$$BIG_{N+1}^\Delta = (BIG_N^\Delta * X_{factor} + \delta X_1) * Y_{factor} + \delta Y_1 \quad (7)$$

The X_{factor} and Y_{factor} make space for their (dX_{min}, dY_{min}) or $(\delta X_1, \delta Y_1)$. The resultant big integer, for the purposes of being transmitted in a text message as part of the WEA is encoded in a string by using a higher base B conversion:

$$\overline{BIG}_B^{\Delta_{min}} = M \bullet BIG_{N+1}^{\Delta_{min}}$$

$$\overline{BIG}_B^\Delta = M \bullet BIG_{N+1}^\Delta$$

Bullet \bullet is the string concatenation operator. The size of M is fixed to ensure lossless decoding. Table 1 shows the different transformations using an example polygon from the NWS corpus.

An estimate of the bounds on BIG can be obtained by noticing that we are essentially placing each dX_i and dY_i , or ΔX_i and ΔY_i into an M sized space, essentially “shifting” by $\log_2(M)$ bits, concatenating into a big number, and then chopping into B sized characters (each $\log_2(B)$ bits). Using two characters to encode M^\P , the approximate length in base B characters is:

$$\text{len}(\overline{BIG}_B^{\Delta_{min}}) \approx 2 + \frac{(\log_2(X_{factor}) + \log_2(Y_{factor}) + 2(N-1)\log_2(M))}{\log_2(B)} \quad (8)$$

^{\S}Since we add one to all deltas in Eq.3, M has to be strictly greater than all $dX_i + 1$ and $dY_i + 1$ such that we can decode correctly.

^{\P}We actually need just $\log_2 M$ bits for M , but we found that a smaller number of bits, n , to choose among 2^n fixed values of M , yields results almost as good

Transformation	Variable	Value
$T^{\Delta_{min}}$	O	{31.35,-85.42 31.27,-85.82 31.43,-85.85 31.6,-85.42 31.35,-85.42}
	O'	{3135,8542,3127,8582,3143,8585,3160,8542,3135,8542}
	$T^{\Delta_{min}}$	{1527,2542,8,0,0,40,16,43,33,0}
	$BIG_{N+1}^{\Delta_{min}}$	118002304535865272542
	$\overline{BIG}_{70}^{\Delta_{min}}$	“hfsEYx0N5(xC”
T^{Δ}	O	{31.35,-85.42 31.27,-85.82 31.43,-85.85 31.6,-85.42 31.35,-85.42}
	O'	{3135,8542,3127,8582,3143,8585,3160,8542,3135,8542}
	T^{Δ}	{1535,2542,15,80,32,6,34,85}
	BIG_N^{Δ}	2954312847725352542
	$\overline{BIG}_{70}^{\Delta}$	“1F13Eq4y‘g*g2”

Table 1: Example of compression using *Bignum* with base $B = 70$

	Compressed length					Compression ratio (%)				
	min.	mean	max.	σ	95 th percentile	min.	mean	max.	σ	95 th percentile
$\overline{BIG}_{70}^{\Delta_{min}}$	9	21.9	61	10	44	11.7	19.0	27.3	2.1	22.4
$\overline{BIG}_{70}^{\Delta}$	9	21	61	10	43	10.4	18.0	25.6	1.8	20.8

Table 2: Results for *Bignum* in base $B = 70$ on NWS corpus

$$\text{len}(\overline{BIG}_B^{\Delta}) \approx 2 + \frac{(\log_2(X_{factor}) + \log_2(Y_{factor}) + 2(N-2)\log_2(M))}{\log_2(B)} \quad (9)$$

Decoding is simply finding the modulus in the reverse order. Specifically, use $dY_{min} = BIG_N^{\Delta_{min}} \bmod Y_{factor}$; then $BIG_{N-1} = \lfloor BIG_N / Y_{factor} \rfloor$, then repeat using X_{factor} to get dX_{min} and use M to get all dX_i s and dY_i s.

5 Comparison and Discussion

For comparison we will consider bits per integer, the most widely-used metric for integer compression. To obtain bits per integer for *Bignum* we sum the number of bits for BIG and M , and divide the sum by the total number of integers in the set $T^{\Delta_{min}}$ or T^{Δ} . For all experiments $X_0 = 1600$; $Y_0 = 6000$; $X_{factor} = 3500$; and $Y_{factor} = 10000$.

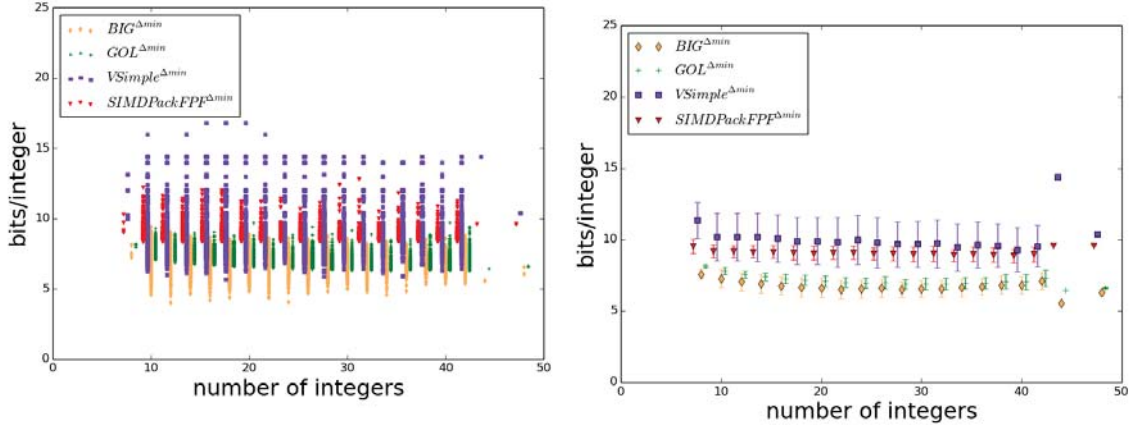
Golomb and Rice coding uses a fixed parameter b to compress a positive integer v via the quotient $\lfloor v/b \rfloor$ output in unary coding and the remainder $v \bmod b$ in binary. For our experiments with the NWS corpus, we found $b = 2^5$ to give the best results.

For comparison with vectorized schemes, we used open source library *TurboPFor*^{||}. For each T^{Δ} and $T^{\Delta_{min}}$ representation we generated a file containing an integer per line, and ran the benchmarking executable^{**}. We only report the best two techniques given by the executable i.e. SIMDPackFPF [13] and VSimple [15].

In figure 6b the mean of bits/integer for *Bignum* is observed to be marginally less than that of Golomb although the increase in mean value with increase in the

^{||}<https://github.com/powturbo/TurboPFor> cloned on October 25, 2015

^{**}executed using: `./icbench <filename>-f1`



(a) Distribution of bits/integer over all compressed polygons grouped by the number of integers (b) Mean bits/integer with error bars for different compression techniques

Figure 6: Experimental comparison of competitive compression techniques for $T^{\Delta min}$ as input. For each polygon, the number of integers on x-axis is equal to $(\#vertices) * 2$. Points were displaced marginally to the left and right on the x-axis to avoid overlapping.

number of integers is similar to that of Golomb. This pattern in mean bits/integer is not seen in figure 7b; higher integer values for T^{Δ} could be a possible reason. Also, Figure 7b may convey that Golomb’s performance was better than *Bignum*, although a one-to-one comparison of bits/integer showed that *Bignum* was better in 5839 instances versus 5531 for Golomb. *Bignum* was better in 10133 instances for $T^{\Delta min}$ representation. Since our primary application is WEA, in table 2 we show the minimum, mean, maximum, standard deviation (σ) and 95th percentile of lengths and compression ratio ^{††} in higher base. In terms of character count, T^{Δ} is more efficient due to an integer pair less.

We also tried other standard string compression algorithms available [16] like *7zip*, and *gzip*, but the compression was not good enough for useful comparison.

6 Conclusions and Future Work

We presented a simple lossless compression technique for list of integers representing a small polygon. We also demonstrated the application of small polygons in a nation-wide emergency service where compression can help target the correct group of people. Furthermore, we show experimental comparison of our technique with other standard compression techniques published.

Future work includes exploring use of statistical skew to improve compression, and a different approach for selection of M . Also, applying *Bignum* for larger set of integers.

^{††}compression ratio = compressed length/decompressed length (O)

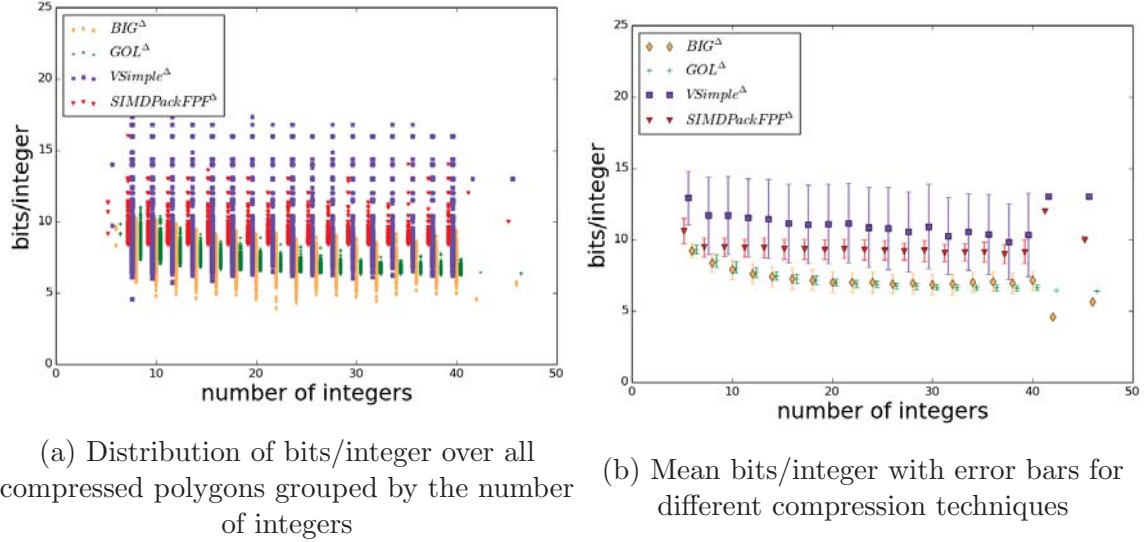


Figure 7: Experimental comparison of compression techniques for T^Δ as input. For each polygon, the number of integers on the x-axis is equal to $(\#vertices - 1) * 2$.

References

- [1] NOAA, “NOAA Public WEA Dataset,” <http://weather.noaa.gov/pub/logs/heapstats/>, 2015, [Online; accessed 25-October-2015].
- [2] H. Erdogmus *et al.*, “Opportunities, options and enhancements for the Wireless Emergency Alert service,” *CMU-SV Publication Number: CMU-SV-15-001*, 2015.
- [3] P. Alliez and C. Gotsman, “Recent advances in compression of 3d meshes,” in *Advances in Multiresolution for Geometric Modelling*. Springer, 2005, pp. 3–26.
- [4] P.-M. Gandoin and O. Devillers, “Progressive lossless compression of arbitrary simplicial complexes,” in *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3. ACM, 2002, pp. 372–379.
- [5] J. Peng, C.-S. Kim, and C.-C. J. Kuo, “Technologies for 3d mesh compression: A survey,” *Journal of Visual Communication and Image Representation*, vol. 16, no. 6, pp. 688–733, 2005.
- [6] A. Maglo, G. Lavoué, F. Dupont, and C. Hudelot, “3d mesh compression: Survey, comparisons, and emerging trends,” *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, p. 44, 2015.
- [7] S. Golomb, “Run-length encodings (corresp.),” *Information Theory, IEEE Transactions on*, vol. 12, no. 3, pp. 399–401, Jul 1966.
- [8] R. G. Gallager and D. C. Van Voorhis, “Optimal source codes for geometrically distributed integer alphabets (corresp.),” *Information Theory, IEEE Transactions on*, vol. 21, no. 2, pp. 228–230, 1975.
- [9] R. F. Rice and J. R. Plaunt, “Adaptive variable-length coding for efficient compression of spacecraft television data,” *Communication Technology, IEEE Transactions on*, vol. 19, no. 6, pp. 889–897, 1971.
- [10] Microsoft, “Microsoft Point Compression Algorithm,” <https://msdn.microsoft.com/en-us/library/jj158958.aspx>, 2015, [Online; accessed 25-October-2015].
- [11] A. Moffat and L. Stuiver, “Binary interpolative coding for effective index compression,” *Information Retrieval*, vol. 3, no. 1, pp. 25–47, 2000.

- [12] D. Lemire and L. Boytsov, “Decoding billions of integers per second through vectorization,” *arXiv preprint arXiv:1209.2137*, 2012.
- [13] D. Lemire, L. Boytsov, and N. Kurz, “Simd compression and the intersection of sorted integers,” *Software: Practice and Experience*, 2015.
- [14] L. T. DeCarlo, “On the meaning and use of kurtosis.” *Psychological methods*, vol. 2, no. 3, p. 292, 1997.
- [15] H. Yan, S. Ding, and T. Suel, “Inverted index compression and query processing with optimized document ordering,” in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 401–410.
- [16] M. Mahoney, “Data Compression Programs,” <http://www.mattmahoney.net/dc/>, 2015, [Online; accessed 26-January-2015].

Appendix A Polygon Transformations

Steps common to $T^{\Delta min}$ and T^{Δ}	
Step 1: Starting with polygon O , round all numbers to 2 (or 3) decimals precision, convert to integers to drop the decimal point, and switch sign of Y_i , so both X_i and Y_i are positive integers, to produce O' : $X_i = int(100 * X_i); Y_i = -int(100 * Y_i)$	
Step 2: Drop the last point N which is a duplicate of the first point since these are closed polygons	
Steps specific to $T^{\Delta min}$	Steps specific to T^{Δ}
Step 3: Compute $X_{min} = \inf_i X_i$, $Y_{min} = \inf_i Y_i$.	Step 3: Compute deltas for coordinates where $i \in [1, N - 2]$:
Step 4: Compute deltas for coordinates such that $i \in [1, N - 1]$: $dX_i = X_i - X_{min}$ $dY_i = Y_i - Y_{min}$, where dX_i and dY_i are non-negative integers.	Step 4: Compute deltas for X_1 and Y_1 from a the chosen “origin”, $(X_0, Y_0) = (1600, 6000)$: $\Delta X_{i+1} = X_{i+1} - X_i$ $\Delta Y_{i+1} = Y_{i+1} - Y_i$
Step 5: Compute deltas for X_{min} and Y_{min} from a chosen “origin”, origin (X_0, Y_0) : $dX_{min} = X_{min} - X_0$ $dY_{min} = Y_{min} - Y_0$ We found (1600, 6000) most effective (see Figures 3a and 3b).	Step 5: Many of the Δ s are negative integers which causes problems for the compression techniques discussed below. Therefore, every ΔX_i or ΔY_i element e will be converted as follows: $e = \begin{cases} 2e, & \text{if } e \geq 0 \\ -2e - 1, & \text{if } e < 0 \end{cases}$
Resultant set: $T^{\Delta min} = \{dX_{min}, dY_{min}, dX_1, dY_1, \dots, dX_{N-1}, dY_{N-1}\}$	Resultant set: $T^{\Delta} = \{\delta X_1, \delta Y_1, \Delta X_2, \Delta Y_2, \dots, \Delta X_{N-1}, \Delta Y_{N-1}\}$

Table 3: Transformation steps applied to get two set of integer representations for each polygon. Both representations are provided as input to all compression techniques.